HpMC: An Energy- Aware Management System for Multi-Level Memory Architectures

HpMC: An Energy-aware Management System of Multi-level Memory Architectures

ChunYi Su†
David Roberts‡

Edgar A. León§
Kirk W. Cameron†
Bronis R. de Supinski§

Gabriel H. Loh‡
Dimitrios S. Nikolopoulos♮

†Virginia Tech
sonicat@vt.edu
cameron@vt.edu

§Lawrence Livermore National Laboratory
leon@llnl.gov
bronis@llnl.gov

‡Advanced Micro Devices, Inc.
gabriel.loh@amd.com
david.roberts@amd.com

♮Queen's University of Belfast
d.nikolopoulos@qub.ac.uk

ABSTRACT

DRAM technology faces density and power challenges to increase capacity because of limitations of physical cell design. To overcome these limitations, system designers are exploring alternative solutions that combine DRAM and emerging NVRAM technologies. Previous work on heterogeneous memories focuses, mainly, on two system designs: PCache, a hierarchical, inclusive memory system, and HRank, a flat, non-inclusive memory system. We demonstrate that neither of these designs can universally achieve high performance and energy efficiency across a suite of HPC workloads. In this work, we investigate the impact of a number of multi-level memory designs on the performance, power, and energy consumption of applications. To achieve this goal and overcome the limited number of available tools to study heterogeneous memories, we created *HMsim*, an infrastructure that enables n-level, heterogeneous memory studies by leveraging existing memory simulators. We, then, propose *HpMC*, a new memory controller design that combines the best aspects of existing management policies to improve performance and energy. Our energy-aware memory management system dynamically switches between PCache and HRank based on the temporal locality of applications. Our results show that HpMC reduces energy consumption from 13% to 45% compared to PCache and HRank, while providing the same bandwidth and higher capacity than a conventional DRAM system.

CCS Concepts

•**Hardware** → **Emerging architectures;** *Platform power issues; Memory and dense storage;* •**Software and its engineering** → Main memory;

MEMSYS'15 October 5–8, 2015, Washington, DC, USA

DOI: 10.475/123_4

Keywords

Energy efficiency; multi-level memories; phase change memory.

1. INTRODUCTION

While memory bandwidth has increased over the years, new challenges have emerged as system designers attempt to increase DRAM capacity to meet the demand of applications within a reasonable power budget. With large-scale, in-memory data analytics now driving the demand for memory capacity, bandwidth, and latency, traditional DRAM technologies are insufficient because of high static power consumption, limited capacitor downscaling and limited bandwidth scaling from single-level designs [23].

Earlier research to overcome DRAM's power limitations [4, 13, 17, 18, 27] proposed heterogeneous memory systems that combine DRAM, for performance, with non-volatile RAM (NVRAM) memory, for power-conscious capacity scaling. NVRAM technologies include phase change memory (PCM) [25] and STT-RAM [2]. These heterogeneous designs include new memory management policies to improve performance and reduce energy consumption, using two fundamental memory organizations (see Figure 1).

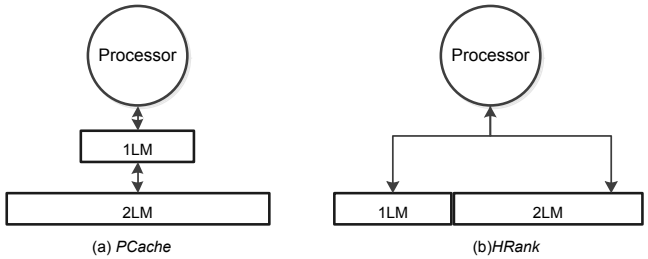

Figure 1: Heterogeneous main memory organizations: (a) *PCache*, a hierarchical, inclusive system, and (b) *HRank*, a flat, exclusive system.

Figure 1(a) shows a hierarchical, inclusive system. The first level of main memory, *1LM*, is used as a cache for the second level, *2LM*. This is similar to existing cache hierar-

chies in current systems but for main memory. The 1LM memory is not visible to the operating system (OS) and is managed by the memory controller (MC). Memory management policies for this design [10, 17] treat 1LM as an associative cache and use LRU replacement to migrate pages. In this paper, we refer to these class of policies as *PCache* or Page Cache for main memory. Note that PCache is different from the *page cache* term used in kernel file systems referring to a memory cache to store recent data from disk.

Figure 1(b) shows a flat, exclusive system. In this design, 1LM and 2LM are exclusive physical memory spaces. The OS manages both memory spaces while the MC supervises page migrations between them. Several policies to migrate pages in this flat design have appeared recently [18, 27]. These policies use the following principles: (1) Place performance-critical pages in 1LM and non-performance-critical pages in 2LM, to achieve a combination of overall high performance and low power dissipation; (2) Rank pages based on history of the number of references and access recency. (3) Periodically migrate pages between 1LM and 2LM based on their ranking history. In this paper, we refer to this type of policies as *HRank* or Historical Ranking.

Although these two memory policies have exhibited promising results, we show that neither policy sustains high performance and, at the same time, low energy consumption across a range of high-performance computing (HPC) workloads. Table 1 shows the memory bandwidth and energy consumption of two HPC applications using these policies (we describe these experiments in Section 4). For pF3D, PCache is better than HRank from an energy consumption point of view. However, for LULESH, HRank is better because it provides almost the same performance as PCache while using less energy. As we will demonstrate later, the performance and energy characteristics of these policies are application dependent. For example, when data reuse is low, PCache may result in excessive data migrations, while HRank can save energy by delaying such migrations and directly accessing the 2LM. These observations motivate our investigation toward more effective management policies of heterogeneous memory systems.

Table 1: Performance and energy of pF3D and LULESH under two memory policies.

Application	Policy	Bandwidth	Energy
pF3D	PCache	7.43 GB/sec	167.6 J
	HRank	11.49 GB/sec	218.0 J
LULESH	PCache	7.91 GB/sec	192.6 J
	HRank	7.99 GB/sec	157.3 J

We faced two significant challenges in this work. First, the number of tools available to study multi-level memories is insufficent to analyze and compare memory management policies for application simulations. And, second, the limited guidance and insights into the impact of heterogenous memories on the performance, power, and energy consumption of applications.

In the first part of this paper, we present a trace-based simulator called *HMsim* to simulate multi-level, heterogeneous memory systems. It leverages well-known simulation components: the AMD SimNow^{TM} simulator[1] for processor simulation and the University of Maryland's DRAMSim [20] for the simulation of each level of memory. We execute an application on the AMD SimNow simulator to generate memory traces for each level of memory and, then, these traces are fed to different instances of DRAMSim to obtain memory performance and energy characteristics. For the second memory level, we re-architected DRAMSim to simulate an emerging PCM memory.

In the second part of the paper, we propose HpMC (Hybrid Policies Memory Controller), a new memory controller design that selectively employs the PCache and HRank policies to deliver better performance and lower energy consumption. HpMC implements a policy switching engine and several new components that extend a single-level MC to facilitate switching policies and migrating pages between 1LM and 2LM. In addition, HpMC implements an energy-aware scheme that periodically analyzes temporal locality based on reuse distance and uses the result as a guide to switch between PCache and HRank for energy optimization. HpMC is a co-designed hardware-software mechanism that manages memory at a page granularity. In the HMsim system, HpMC is implemented in the AMD SimNow simulator to interface the two memory levels simulated by DRAMSim.

We use pF3D and LULESH, two HPC applications of interest to the U.S. Department of Energy, to understand how PCache and HRank impact performance and energy. We also analyze the spatial and temporal locality of numerous, diverse memory access patterns collected from the CORAL benchmarks[2] and lmbench [11]. These experiments guide the design of HpMC, our energy-aware memory controller. HpMC reduces energy consumption from 13% to 45% on our suite of applications compared to well-know two-level management policies, while providing almost the same bandwidth and larger capacity than a DRAM-only system.

The contributions of this work are as follows:

- A heterogeneous memory simulator designed to analyze memory performance and energy in a two-level memory system. It couples fast processor simulation with detailed memory simulation. We validate this infrastructure against two real compute systems.
- A memory controller design to enable switching between memory policies to deliver high performance and energy efficiency. Our empirical findings suggest that no single policy delivers the best performance and energy consumption for a range of HPC workloads.
- A new energy-aware scheme that dynamically switches between memory management policies to optimize for energy-efficiency based on the temporal locality of applications.
- Temporal locality of applications is an effective metric to guide a two-level memory management system for performance and energy considerations.

2. HPMC DESIGN

Figure 2 shows a block diagram of the major components of our hybrid policies memory controller, HpMC. HpMC is designed to process read and write requests from the last-level cache (LLC) and route requests to specific memory

[1]http://developer.amd.com/tools-and-sdks/cpu-development/simnow-simulator/

[2]https://asc.llnl.gov/CORAL-benchmarks/

layers according to the active management policy. HpMC implements the PCache and HRank baseline policies and a *Policy Switching Engine* (PSE) to switch between them. PSE manages frames and notifies the OS when physical pages are migrated between the two memories. We begin describing the main implementation details of HRank and PCache. Later in this section, we describe new components that facilitate dynamic switching between these two.

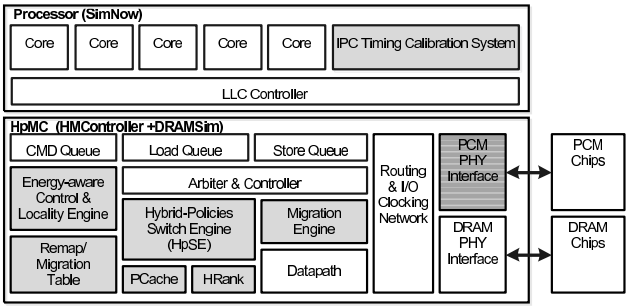

Figure 2: HpMC design.

2.1 Heterogeneous Memories Policies

2.1.1 PCache

PCache manages memory in a hierarchical, inclusive memory system. Our implementation, which is based on that of Qureshi et al. [17], uses DRAM as a cache for PCM. The OS manages the PCM space while the MC manages the DRAM without OS involvement. The DRAM is implemented as an associative cache with an LRU replacement policy. On a DRAM miss, the PCM frame that contains the cache line is fetched into DRAM. The MC uses an inclusion bit to indicate whether a frame holds a copy in PCM and eight bits to track dirty sub-blocks of a frame. PCache adopts a lazy write-back strategy to reduce PCM write operations. With this strategy, when a frame is evicted from DRAM, the write-back operation only happens when the inclusion bit is set to zero or some dirty bit is set to one. PCache leverages a *Remap/Migration Table* in the MC to track the mapping between PCM frame IDs and DRAM frame IDs. When a memory request arrives, the MC checks the *Remap/Migration Table* to see if the requested frame is cached in DRAM. It also supports the line-level writes technique, where the MC only writes dirty sub-blocks back to PCM to reduce traffic.

2.1.2 HRank

Our HRank implementation manages a flat, exclusive memory system based on hot-cold frames [27, 18]. Unlike previous work that tracks recent and adjacent accesses with a multiple-queue (MQ) ranking system [28], we rank frames according to their number of references. The policy selects the top-N hottest frames from the ranking list every 10 ms epoch and moves them to DRAM, while keeping the other frames in PCM. It maintains hot and cold lists to track hot and cold frames. HRank compares the new and previous ranking results (in the hot and cold lists) to determine if migrations to DRAM are necessary. If no frames are moved, the MC uses the old hot and cold lists for the new epoch. Otherwise, it schedules the migration of frames in the *Migration Engine* queue. The HRank algorithm is simple but effective. It simplifies the design of the memory controller because it only ranks and migrates frames every 10 ms. In contrast, MQ-based algorithms update the entire MQ ranking system possibly migrating pages on every memory reference. With HRank, the MC periodically updates the migration history in the OS' *Remap/Migration Table* to keep the system consistent.

2.2 Policy Switching Engine

HRank and PCache are fundamentally different: PCache is an inclusive system, while HRank is an exclusive system. When the PSE switches from one policy to another, it must ensure that the OS is aware and consistent with changes from inclusive to exclusive and vice versa. When these changes occur, the goal of the memory controller is to reduce data (frame) migrations as much as possible.

Several steps are performed to switch from PCache to HRank. The general idea is to keep the data residing in DRAM in DRAM and mark the associated frames in PCM free. 1) The PSE interrupts the CPU and notifies the OS to update the page table entries (PTEs). 2) The OS replaces the old PCM frame IDs with new DRAM frame IDs in PTEs and flushes the corresponding TLB entries according to the *Remap/Migration Table*. 3) The PSE frees the PCM frames stored in the *Remap/Migration Table*, because HRank does not maintain duplicates in the PCM space. 4) The PSE cleans up the information in the *Remap/Migration Table* and begins to track the migration history.

Similarly, the following steps are taken to switch from HRank to PCache. 1) The PSE notifies the *Migration Engine* to cancel any scheduled migrations and cleans up the *Remap/Migration Table*. 2) To restore the inclusion property the PSE checks the hot list for DRAM frame IDs and allocates unused frames in the PCM to restore the <DRAM, PCM> mapping in the *Remap/Migration Table*. If PCM does not have enough unused space to restore the inclusion property, the PSE must vacate the least frequently used PCM frames. The PSE then moves the vacated frames to a removal list. 3) The PSE notifies the OS to replace old DRAM frames with new PCM frames in the PTEs based on the new *Remap/Migration Table*. When the PSE sends the removal list information to the OS, the OS invalidates the corresponding PTEs in the page table, flushes TLB entries, and programs the DMA engine to write dirty frames to a system-reserved buffer in the PCM memory to avoid spilling to storage. Thus, switching policies does not trigger accesses to secondary storage avoiding significant performance penalties.

2.3 Remapping / Migration Table

The *Remapping/Migration Table* is used in both PCache and HRank. Each entry in the table has two columns to record frame IDs and several bits. In PCache policy, the two columns track the mapping between DRAM frame IDs and PCM frame IDs. In HRank, the two columns track the migration history. The first column records the source frame IDs and the second column records the destination frame IDs. The MC periodically updates the migration history in the OS to keep the system consistent.

Each entry has an inclusion bit and 8 dirty bits used in the PCache policy as described before. In addition, we leverage the idea from Ramos *et. al.* [18] that uses two additional bits in the *Remapping/Migration table* for the communica-

tion between the MC and the OS. When the first one, the *Migrating* bit, is set, the frame is currently in migration status. The OS sets the second, the *Replacing* bit, when the OS is replacing the content of the frame. The OS controls the update of *Replacing* bit, and the MC controls the update of *Migrating* bit. To maintain the system robustness, the *Replacing* and *Migrating* bits are exclusive and cannot be set at the same time. The OS and the MC check whether the other bit is set before they update the one that they control. Since the *Remap/Migration Table* is maintained by both the MC and the OS, they use a memory-mapped register in the MC as an atomic operation token.

2.4 Migration Engine

The *Migration engine* uses a queue to record the scheduled migrations. It processes migrations sequentially. In each migration, it reads the source frame into a buffer and sets the *Migrating* bit to 1. Once the *Migrating* bit is set, it writes the content of the buffer to the new destination and resets the *Migrating* bit when the migration finishes. When a memory request arrives, it checks the *Migrating* bit to see if the frame is undergoing transfer. During the migration, if the memory request is a READ, it reads the data from the source frame; if the request is a WRITE, the *Migration Engine* cancels the migration, finishes the write operation, and inserts the migration in to the queue again. The MC uses a counter to monitor the frequency of the cancellation. If cancellation were to occur too often, the MC halts the migration engine and resumes it in the next epoch.

2.5 Energy-aware Controller

The Energy-aware Controller (EaC) periodically uses a *locality engine* to track the reuse distance distribution among memory references and calculate a *degree of temporal locality* denoted as M_t, which we define in Section 4. The locality engine uses 16 64-bit counters to track the distribution and estimate M_t. Each counter i stores the number of memory references with reuse distance between 2^i to 2^{i+1}. EaC switches between PCache and HRank policies to optimize energy based on M_t. We shall demonstrate later that the energy consumption of both policies has a strong correlation to M_t. Reuse distance analysis is a favorable tool for predicting locality and performance. However, prior research [7] has shown that reuse distance analysis incurs a high performance penalty in cache systems. We argue that the overhead of reuse distance in main memory is negligible and feasible for online estimation for the following two reasons: 1) The traffic in main memory is 40-100× smaller than in the cache system. Thus, the estimation overhead can be drastically reduced. 2) Several stochastic models have been proposed to approximate the reuse distance online with small computation cost. Shen et al. [21] proposed a small hardware analysis device with a stochastic model that maps cheaper time distance to a more expensive reuse distance with a 1% prediction error and negligible computing overhead (3-8 μs).

2.6 Storage Overhead

A key challenge for enabling high-performance heterogeneous memories is to design a cost-effective metadata system (e.g., *Remapping/Migration table*) at a fine granularity. Table 2 shows the storage overhead incurred by the HpMC on a heterogeneous memory system with 1 GB DRAM + 8 GB PCM capacity. HRank needs 9 MB in total to maintain the hot, cold and ranking lists. Because PCache only uses the *Remapping/Migration Table* for page migration without any additional components, PCache is set to 0 in Table 2 to avoid counting the *Remapping/Migration Table* space twice. In addition, each entry in the *Remapping/Migration table* requires 55 bits for storing tags (22×2 bits, 2 column frame IDs, 1 inclusion bit, 8 dirty bits, 1 migrating bit, 1 replacing bit) and 256 K entries. The total size of the *Remapping/Migration* table is 1.72 MB. The total queue size of the *Migration Engine* is 1.5 MB (44bits for source and destination frame IDs × 256K queue size). Lastly, the locality engine needs 128 bytes (16 counters × 64 bits) to track the reuse distance distribution in our implementation. The total storage overhead for a typical 1 GB DRAM + 8 GB PCM two-level memory is 12.23 MB. For fast access, we assume that the storage in all the components in the MC is made of SRAM. The choice of SRAM is a simplification for our study. Recent work proposes new designs using die-stacked DRAM to improve metadata systems performance [10].

Table 2: Storage overhead of HpMC.

Component	Storage overhead
HRank	9 MB 16 bits ref. counter * (256 K hot list + 2 M cold list + 2.25 M ranking list)
PCache	0MB
Remap / Migration Table	1.72 MB 55 bits tags * 256 K entries
Migration Engine	1.5 MB 44 bits * 256 K entries
Locality Engine	128 bytes 16 counters * 64 bits
Total Size	12.23 MB

2.7 Switching Overhead

The PSE switching overhead of the HpMC includes a TLB flush, updating the *Remap/Migration Table*, and updating PTEs in the OS. The TLB flush and *Remap/Migration Table* update are done by hardware. However, the PTEs update is expensive in a commodity OS. The overhead is significantly smaller, however, in lightweight kernels commonly used in HPC [19, 5]. In these systems, the OS does not use multi-level page tables and keeps the virtual and physical address mapping simple. Although we do not explicitly model the switching latency overhead for the PTEs updates, we summarize the switching rate of the applications and benchmarks used in this work in Table 3. The average switching percentage across all the benchmarks is low at 5.6%, i.e., when the system checks to determine if a policy switch is necessary, only 5.6% of the times, in average, there is an actual switch. Thus, even though the switching overhead can be significant, the frequency of this operation is low. In addition, the memory controller can mitigate the switching overhead by lengthening the interval of checking whether a policy switch is necessary.

3. SIMULATION APPARATUS

We built a trace-based simulation apparatus, HMsim, to evaluate our hybrid memory controller on a two-level memory system with DRAM and PCM memories. HMsim uses the AMD SimNow simulator for processor simulation and

Table 3: PSE switching percentage over the complete execution of a benchmark.

Program	%	Program	%
CNS.STENCIL	1.3	UMTmk	1.5
Graph500	1	MILCmk	0
MEM.BW	1.2	LULESH	26.5
MEM.LAT	1.1	AMGmk	1.1
pF3D	17.5		

Table 4: PCM and DRAM Characteristics.

Performance	PCM	DRAM	Ratio	Range
READ latency	55 ns	15 ns	3.7	3-6
WRITE latency	90 ns	15 ns	6.0	5-30
READ energy	3.56 pJ/bit	1.04 pJ/bit	3.4	2-8
WRITE energy	12.35 pJ/bit	0.35 pJ/bit	35.5	10-100

DRAMsim for memory simulation. Both simulators are needed because the former is a functional simulator and does not provide a memory performance model. Coupling DRAMsim with the AMD SimNow simulator allows us to obtain cycle-accurate memory simulations from the memory traces obtained by running an application on the AMD SimNow simulator. We use two instances of DRAMsim, one for DRAM and one for PCM.

We developed a heterogenous memory controller, HMController, within the AMD SimNow simulator to simulate and manage access to a two-level memory system. The HMController consists of the controller blocks discussed in the previous section as part of the HpMC. The simulation has two steps. In the first step, HMsim uses the HMController to collect memory traces of DRAM and PCM memories based on a specific memory policy. In the second step, HMsim feeds the memory traces to DRAMsim to obtain performance and energy results.

HMsim uses two instances of DRAMsim, one to simulate the DRAM and another the PCM. For the latter, we developed a DRAMsim configuration that simulates a PCM architecture, which adopts the buffer reorganization [8] and data-comparison write (DCW) [26] techniques that improve PCM write latency and energy. In addition, we assumed that future PCM systems will support a PreSET-like mechanism [15] in which the write latency is effectively the faster *RESET* latency instead of the slower *SET* latency[3].

We summarize the performance characteristics of our PCM and DRAM systems in Table 4 and also compare the performance numbers with recent literature [12, 8]. The *Ratio* column shows the ratio of PCM to DRAM performance in terms of latency and energy. The *Range* column shows the range of the ratio based on a literature survey. Our PCM to DRAM rations fall within those presented in the literature.

The main architectural characteristics of the simulated system are listed in Table 5. HMsim uses these characteristics to build the performance and energy models. The details of the models can be found in a Micron technical report [1]. In this work, HMsim simulates a four-core, out-of-order processor equipped with an 8 MB, 2-way instruction and data cache. Unless specified otherwise, the memory system configuration consists of a two-level, heterogeneous memory of 1 GB DRAM and 8 GB PCM. It has four DRAM channels and four PCM channels; each channel has two DIMM ranks. In all simulations, the simulator assumes no cold page faults and that all data is placed initially in 2LM.

[3]Removing this assumption would result in the SET operation being up to 8X the read operation.

Table 5: Characteristics of the Simulated System.

Feature		Value/Configuration	
Processor			
Processors (800MHz, x86-ISA)		4-way out-of-order processor	
I/D Cache		2-way, 128M lines, 64 bytes	
TLB		128-entries	
Cache block size/ page size		64 bytes/4KB	
Memory Systems			
Memory Controller		1333MHz, 4 channels, 8KB row size, close page	
Memory Devices (8x width, 1.5V)		**DRAM**	**PCM**
Delay	tRCD	15ns	55ns
	tRAS	36ns	71 ns
	tRC	51ns	126 ns
	tRP	15ns	90 ns
Current	Idd0	130mA	240mA
	Idd 2N	40mA	40mA
	Idd 3N	62mA	62mA
	Refresh	240mA	0mA

Before we can use HMsim for performance evaluation, we need a mechanism to calibrate the timing system in the AMD SimNow simulator, an x86 dynamically-translating, instruction-level platform simulator. To enable fast simulation, the simulator abstracts away the timing accurate feature of the entire computer system. Although HMsim can leverage fast simulation in the AMD SimNow simulator to explore applications at scale, timing accuracy remains the major challenge. The basic timing unit of the simulator is an instruction; all instructions are assumed to execute in the same amount of time and are one clock cycle in length. This assumption may overlook the performance impact of long latency events such as cache, memory accesses, and page faults if they are not overlapped with other instructions. Fortunately, the AMD SimNow simulator provides an interface to set the IPC (number of instructions per cycle) for each application. In the next section, we discuss our IPC calibration model to improve the accuracy of the timing system and then validate the calibrated simulation against real hardware using two modern compute systems.

3.1 IPC Calibration Model

We propose an IPC calibration model to calibrate the AMD SimNow simulator timing mechanism. The goal of the model is to predict a proper IPC value for each application, so as to generate the same demand for bandwidth from the simulated LLC controller and the native systems. The model predicts the IPC of an application by using input from native execution. The input includes the native measured IPC and a set of hardware event rates $(e_1, e_2, ..., e_n)$. We select events that are critical to system performance, including the memory controller reads and writes, L1, L2, L3 hits, floating point instructions, branch instructions, and TLB misses. All selected events can be found in most contemporary processors. Each event rate, e_i, is the number of occurrences of event i divided by the number of elapsed processor cycles during the execution. We model the simulator IPC as a linear function of the native IPC and event rates:

$$Sim\ IPC = (Native\ IPC) * \alpha_0 + \sum_{i=1}^{n} (\alpha_i * e_i) \qquad (1)$$

We trained the model in Equation 1 with event coefficients $a_i, i = 0, ..., n$ by using multivariate regression. We first collected the IPC, event rates, and bandwidth from benchmarks as training samples from native machines. We then ran the same benchmark on the HMsim using a single-level memory, DRAM, and manually selected the AMD SimNow simulator's IPC value that generates the same amount of bandwidth from the LLC Controller as the bandwidth collected in native machines to be the prediction target.

3.2 Validation Against Native Systems

We validate the performance and power obtained with HMsim by applying the IPC calibration model against two real computing systems. The first system is a single-socket, 8-way Intel Westmere processor with an integrated memory controller that supports 3 memory channels of DDR3 DIMM; each DIMM has 2 GB memory capacity. The second system is an 8-way Intel Haswell processor with dual-channel DDR3 memory and 8 GB capacity.

Since these existing systems have only one level of memory (DRAM), we configure HMsim as a single-level memory, DRAM system and validate it using three benchmarks: MEM.BW and MEM.LAT from lmbench and AMGmk from the CORAL benchmark suite. We compare the bandwidth, latency, and power of the simulated system with the corresponding measurements in the two real systems. We validate the memory bandwidth and latency against the Westmere machine. But since we do not have access to the power measurement infrastructure on the Westmere, we use the newer Haswell system for the memory power validation using RAPL [3]. The results are discussed below.

3.2.1 Bandwidth

We ran MEM.BW to compare the relative bandwidth changes in eight operation modes: rd, wr, cp, frd, fwr, fcp, bzero, bcopy, rdwr. We used the LIKWID tool [22] to measure memory bandwidth on the Westmere platform. The top left graph in Figure 3 shows the normalized bandwidth from HMsim and the Westmere system. The Westmere and HMsim results are normalized to their respective bandwidth obtained with MEM.BW.RD. The results show a small relative difference between real and simulated system with an average error of 6.1%.

3.2.2 Latency

We used the MEM.LAT benchmark from lmbench and varied stride sizes from 64 to 4096 to validate the latency. The top right graph in Figure 3 shows the simulated and native measured latencies. We normalized all simulated and native results to their respective stride 64 (MEM.LAT.64). In both simulation and native results, we observe that latency is reduced when the stride size increases. This is because when the MEM.LAT benchmark traverses the same amount of data, larger strides access the memory system less frequently than smaller strides and alleviate the waiting time in the transaction queue of the MC. Although the degree of degradation is slightly different, we can see that HMsim can accurately capture the trend of degradation.

3.2.3 Power

The bottom graph in Figure 3 shows the normalized power using MEM.BW, MEM.LAT, and AMGmk. Results are normalized to the power consumed by MEM.LAT stride 64. We ran AMGmk with three input sizes: 50, 100, and 200. We measured the native DRAM power using RAPL on the Haswell machine. It is worth mentioning that the Haswell processor uses an FIVR (fully integrated voltage regulator) to control voltage and frequency on a per-core basis and the uncore DRAM. RAPL can read FIVR voltage directly and get power values via MSRs (model-specific registers). The Haswell architecture no longer uses the modeling approach used in earlier processor generations. DRAMSim, on the other hand, uses elapsed cycles and currents of different CAS commands to model memory power. Although the two systems use different approaches to derive power, the power values in HMsim still capture the same trend as measured in the Haswell system.

To summarize, our validation experiments show that, although HMsim does not simulate the identical cycle-by-cycle behavior to that of a native system, it remains sufficiently accurate for evaluating our HpMC memory controller. In particular, we are interested in the relative performance and power tradeoffs introduced by different management policies and, as we demonstrated in this section, HMsim captures the same trends in performance and power as those in real systems.

4. PCACHE AND HRANK PERFORMANCE AND LOCALITY ANALYSIS

In this section, we analyze the effective processor bandwidth and energy consumption of the HpMC using PCache and HRank. Our analysis include how spatial and temporal localities affect energy consumption of both policies and use these results to build the switching governor for the EaC to optimize energy consumption. We focus our analysis on pF3D and LULESH.

4.1 Metrics

We first define metrics to quantify performance and locality of the workloads studied in this section.

4.1.1 Effective Processor Bandwidth

The memory bandwidth we estimate from DRAM and PCM includes the migration traffic between them. From the processor's point of view, migration traffic is off the critical path. Thus, we need to exclude migration traffic to get the effective processor bandwidth. We define the effective processor bandwidth of PCache in Equation 2.

$$Effective\ BW_{PCache} = BW_{DRAM} - BW_{PCM} \quad (2)$$

BW_{DRAM} and BW_{PCM} are the bandwidth we measured from DRAM and PCM systems respectively. Because the processor only accesses the DRAM system and BW_{DRAM} includes migration traffic, the effective processor bandwidth of the PCache policy is BW_{DRAM} subtracted by BW_{PCM} (i.e. BW_{PCM} is all used for migration).

In HRank, because the processor can access both DRAM and PCM, we need to sum up the bandwidth of both memories. We also need to exclude the bandwidth due to migrations. The adjusted effective processor bandwidth of the HRank policy is defined in Equation 3. λ_{DRAM} and λ_{PCM} represent the total data traffic between the processor and the two memory levels and $\lambda_{Migration}$ represents the total

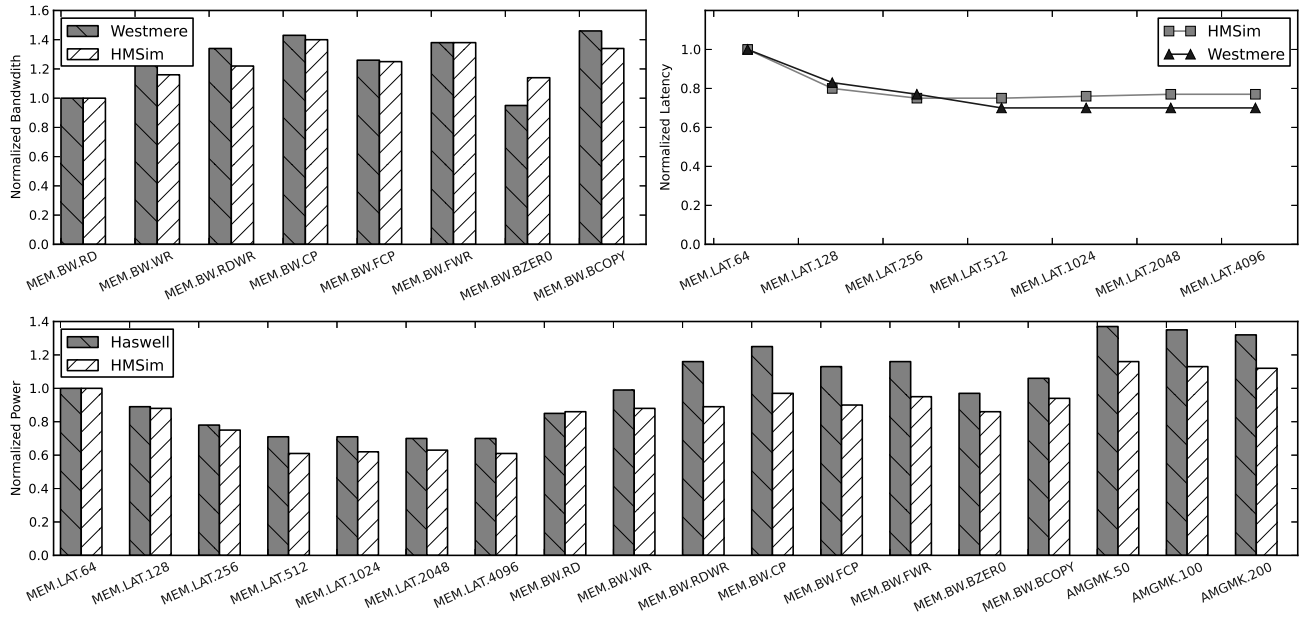

Figure 3: Comparing HMsim memory bandwidth, power, and latency against two native systems.

migration traffic between DRAM and PCM.

$$Effective\ BW_{HRank} = (BW_{DRAM} + BW_{PCM}) * \frac{\lambda_{DRAM} + \lambda_{PCM}}{\lambda_{DRAM} + \lambda_{PCM} + \lambda_{Migration}} \quad (3)$$

4.1.2 *Spatial and Temporal Locality*

We use metrics proposed by Weinberg et al. [24] to quantify the spatial and temporal locality of an application. The spatial locality estimates strides between pages, not cache blocks, and is defined in Equation 4. $stride_i$ denotes the fraction of total memory accesses that are of page stride length $i-1$. An application that has all stride-0 references receives a score of 1; an application whose memory references have half page stride 0 and half page stride 1 is evaluated .75, and so forth.

$$M_s = \sum_{i=1}^{\infty} \frac{stride_i}{i} \quad (4)$$

In addition, we quantify the temporal locality using the metric in equation 5. The metric is based on the notion of the distance of data reuse. The reuse distance of memory references to an address A is the number of memory references between two references to A. In equation 5, N denotes the longest reuse distance we traced (N= 2^{16} in our study); $reuse_i$ is the temporal reuse function and represents the fractions of memory references with reuse distance less than or equal to i.

$$M_t = \frac{\sum_0^{log(N)-1}((reuse_{2^{i+1}} - reuse_{2^i}) * log_2 N - i)}{log_2 N} \quad (5)$$

The M_t and M_s scores range between [0,1]. Higher scores mean better locality than lower scores.

4.2 pF3D

Figure 4 shows the pF3D DRAM and PCM bandwidth during execution using PCache and HRank policies. In PCache, the average bandwidth of the DRAM and PCM is 8.17GB/sec and 0.76GB/sec. These values include the migration traffic between the DRAM and PCM. Based on the Equation 2, the effective processor bandwidth is 7.43 GB/sec for pF3D (i.e., 8.17-0.76).

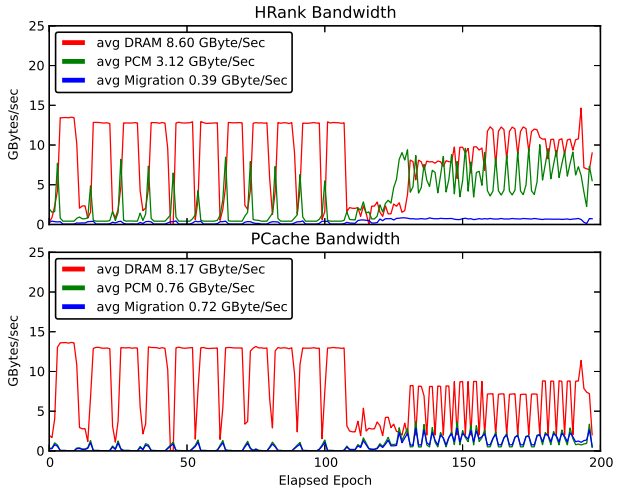

Figure 4: pF3D DRAM and PCM bandwidth using HRank and PCache policies in the HpMC

On the other hand, in HRank policy, the average bandwidth of the DRAM and PCM is 8.6GB/sec and 3.12GB/sec, respectively. The effective processor bandwidth is 11.49 GB/sec calculated using Equation 3 and data traffic in Table 6 (i.e. $(8.6 + 3.12) * \frac{(71.92+19.91)}{(71.92+19.91+1.53)}$). In the pF3D

case, HRank provides more bandwidth than PCache because HRank allows the processor to access the PCM directly.

Table 6: Total data traffic of pF3D and LULESH using the PCache and HRank policies.

	Policy	λ_{DRAM}	λ_{PCM}	$\lambda_{Migration}$
pF3D	PCache	91.87GB	0GB	2.83GB
	HRank	71.92GB	19.91GB	1.53GB
LULESH	PCache	38.17GB	0GB	24.49GB
	HRank	18.23GB	20.18GB	2.26GB

From an energy perspective, our simulation estimates that PCache consumes 167.6J energy during the execution and HRank consumes 218J (see Table 1). PCache use 24% less energy than HRank. When analyzing the data traffic of the two policies in Table 6, we found that, in PCache, the processor accessed 91.81GB of data from DRAM and only 2.83 GB of data are migrated from PCM due to cache misses. The results show that pF3D has a high hit rate with PCache. In contrast, the processor accessed 71.92 GB of data from DRAM and 19.91GB of data from PCM when using the HRank policy. Compared to PCache, HRank causes 7.1x more data accesses in the PCM. Thus, the energy consumption of HRank is more than the PCache in the pF3D case.

4.3 LULESH

Figure 5 shows the bandwidth of LULESH using two policies. Based on equations 2 and 3 and Table 6, HRank provides 7.99GB/sec effective processor bandwidth while PCache delivers 7.91GB/sec. Both policies support almost the same effective processor bandwidth in the LULESH case. However, from an energy perspective, HRank consumes 20% less energy than PCache (157.3J vs. 192.6J). To understand why HRank consumes less energy, we analyze the total data traffic of the two policies. PCache needs to migrate 24.49 GB of data between the DRAM and PCM to meet application demand (i.e. 38.17GB). We observed that LULESH has a low hit rate when using PCache. The total data traffic in the memory system is 38.17+24.49 GB, but only 38.17GB of data is used by the processor. Thus, up to 39% ($\frac{24.49}{38.17+24.49}$) of the total memory traffic is due to data migration between DRAM and PCM. On the other hand, in the pF3D case, PCache only needs to transfer 2.83GB of data between the DRAM and PCM to meet the total 91.81GB processor demand. The effective data traffic in this case is 97%.

In conclusion, we find that PCache minimizes energy in workloads with a high DRAM hit rate and HRank minimizes energy in workloads with a low DRAM hit rate. When the DRAM hit rate is low, PCache with LRU replacement becomes too aggressive in migrating data between DRAM and PCM and wastes energy due to excessive migrations. In contrast, HRank delays migrations and migrates frames only periodically. The periodic migration strategy in HRank conserves more energy when the DRAM hit rate becomes low. We also find that HRank provides extra bandwidth due to direct access to PCM, but this could potentially consume more energy. Our observations generalize to other workloads in the paper.

4.4 Locality Analysis for Energy Optimization

Based on the above observation, we can leverage the locality properties of an application to intelligently select the

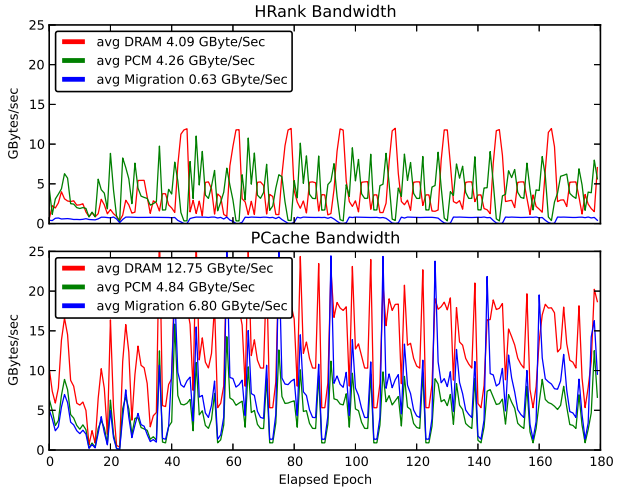

Figure 5: LULESH DRAM and PCM bandwidth using HRank and PCache policies in the HpMC

policy that delivers the lowest energy consumption. The hit rate is decided by two factors: access adjacency (spatial locality) and recency (temporal locality).

To investigate the correlation of locality and energy consumption of two policies, we built a 2D (M_t,M_s) locality map for each benchmark listed in Table 7. Due to space limitations, we only show the results of pF3D, LULESH, AMGmk, MILCmk, Graph500 and UMTmk in Figure 6. The x-axis of each map represents M_t and the y-axis represents M_s. A circle at position (x,y) on the map represents a small period (10 ms epoch) of execution in an application with $M_t = x$ and $M_s = y$ estimated from memory traces of the period. We also estimated the energy consumption of the epoch using PCache and HRank policies and chose the lowest energy policy as the winning policy. If the PCache wins in an epoch with locality scores $M_t = x$ and $M_s = y$, the map plots a green circle at position (x,y) on the map; if HRank wins, the map plots a blue circle. The size of the circle is used to represent the ratio of energy consumption of the losing policy to the winning policy. The bigger the circle, the more energy is saved by the winning policy over the losing one. We analyzed diverse memory patterns from over 3,000 epochs from the aforementioned benchmarks. In Figure 6, we see PCache wins for most epochs in pF3D, MILCmk and UMTmk. Because the circles in the MILCmk and UMTmk are small, there is not much energy difference between the two policies. We also observed that blue circles dominate (i.e. HRank is better) in the LULESH case, even though some green circles are clustered on the top-right (high M_t, M_s values).

In addition, the two rightmost charts show statistical histograms of occurrences of M_s and M_t values from the winning policy for all epochs, in all applications. In the M_s histogram, we observe that both policies span the range from 0.1 to 1 with an irregular distribution. This result indicates that M_s is not a good indicator for selecting the winning policy. This is due to the fact that spatial locality might be broken by the multi-core, out-of-order, parallel execution. In contrast, the M_t histogram shows that the PCache policy prefers higher M_t (greater than 0.65) while the HRank

Table 7: Our suite of applications and benchmarks.

Program	Description	Source
CNS.STENCIL	Simple stencil-based code for computing the hyperbolic components	ExaCT Co-Design Center[4]
UMTmk	Microkernel performing 3D, nonlinear, radiation transport calculation	CORAL Benchmark
Graph500	Scalable data generator and a BFS search kernel	CORAL Benchmark
MILCmk	Microkernel for the MIMD Lattice Computation (MILC)	CORAL Benchmark
AMGmk	Microkernel for a parallel algebraic multigrid solver for linear systems	CORAL Benchmark
LULESH	Proxy-app for shock hydrodynamics	CORAL Benchmark
pF3D	Kernels for simulating laser-plasma interactions	LLNL NIF
MEM.BW	Benchmark from lmbench to measure memory bandwidth	lmbench [11]
MEM.LAT	Benchmark from lmbench to measure memory latency	lmbench

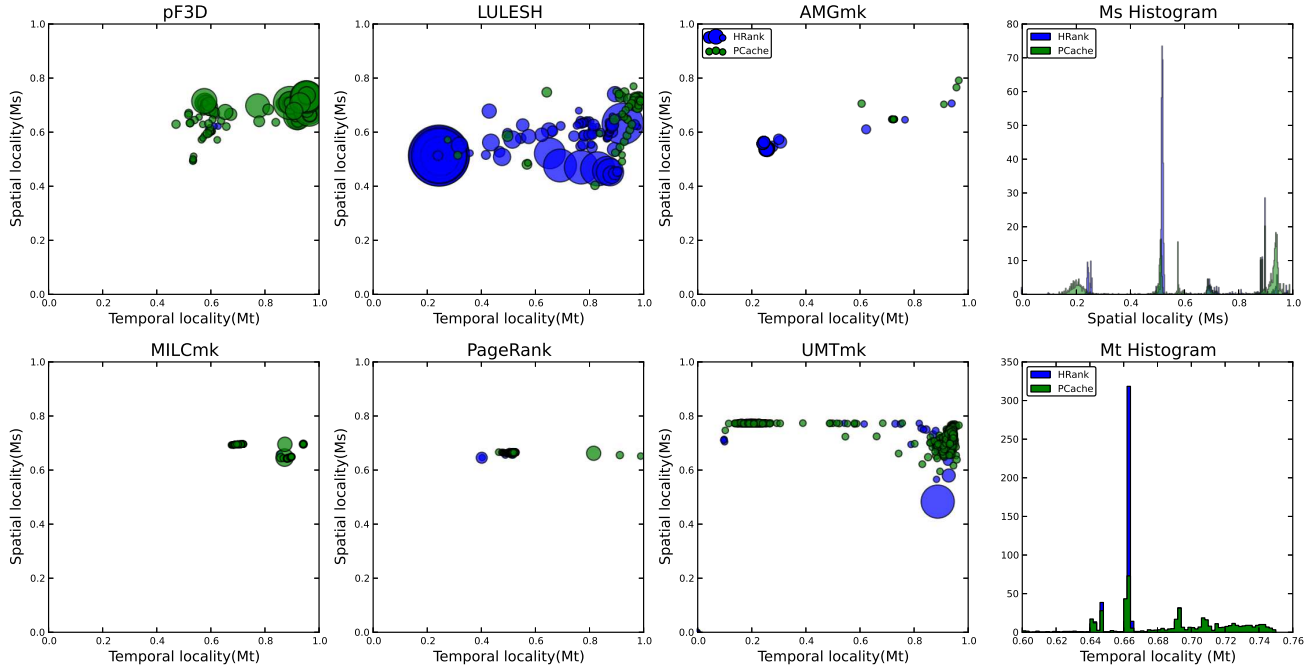

Figure 6: The correlation between locality and energy consumption of HRank and PCache policies

system prefers lower M_t (less than 0.65). The observation also aligns with the nature of the PCache and HRank system designs. When the M_t is high, the data in a page is to be reused again soon, and the page has a higher chance to stay in DRAM without being evicted. In contrast, when M_t becomes low, the reuse of a page in DRAM becomes low, and the page has a higher chance to be evicted in the PCache policy, resulting in more energy consumed by migration. In this case, HRank can reduce energy consumption by less frequent data migrations and by letting the processor access the PCM directly.

5. ENERGY-AWARE HYBRID POLICY

We now show the energy consumption of the EaC mode in the HpMC. EaC controls the dynamic switching between PCache and HRank policies. The EaC periodically checks the temporal locality, M_t, and decides if it needs to switch to a different policy. EaC sets the switching period to be 10ms and uses the locality engine to calculate M_t. Based on the M_t histograms obtained from our benchmarks, we build a switching rule as follows: if $M_t \geq 0.65$, EaC switches to PCache. If $M_t < 0.65$, the EaC switches to HRank.

Figure 7 illustrates the energy consumption of PCache, HRank, and the EaC of pF3D and LULESH. The results indicate that the hybrid approach using EaC intelligently selects the low energy system over time with negligible prediction error. It improves energy consumption by 23% ($\frac{169.3J}{218J}$) in pF3D and 20% ($\frac{155.1J}{192.6J}$) in LULESH, compared with the worst case scenarios and has almost the same energy consumption in both pF3D (169.3 J vs. 167.6 J) and LULESH (155.1 J vs. 157.3 J), compared with the best case scenarios.

5.1 Performance Evaluation

We evaluate the performance of the HpMC using the following policies: PCache, HRank, and EaC. We compare HpMC performance with a DRAM-only system and a PCM-only system. The DRAM system uses 64GB, single-level, 4-channel, DDR3-1333 memory. The PCM system also uses a single-level with 64GB capacity. We select 64GB for the DRAM and PCM systems as the base memory capacity be-

[4]http://exactcodesign.org

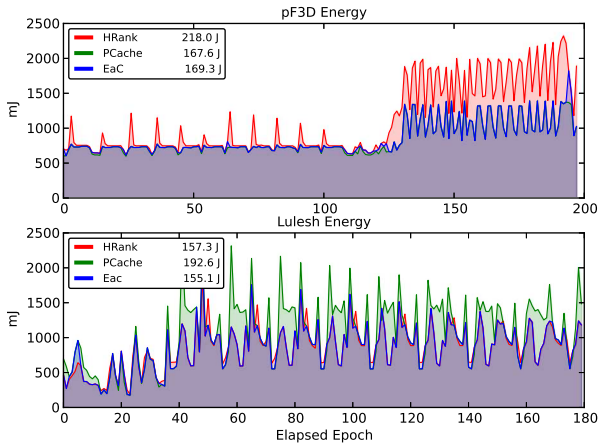

Figure 7: Energy consumption of pF3D and LULESH with PCache, HRank, and EaC.

cause this represents a common setting for state-of-art HPC systems [9]. Qureshi et al. [17] suggested that 1:32 capacity ratio of DRAM to PCM can achieve near DRAM-only performance while conserving more energy. In this study, we select a higher 1:8, DRAM to PCM ratio and use 8GB of DRAM and 64GB of PCM in the HpMC configuration. We use the benchmarks listed in Table 7 to evaluate performance and the results are illustrated in Figure 8. MEM.LAT is not shown because its results are similar to MEM.BW. We sorted the benchmarks based on DRAM system bandwidth from left to right.

The top chart in Figure 8 shows the effective processor bandwidth calculated using equations 2 and 3. We normalized the bandwidth to the DRAM system. The MEM.BW benchmark is used to measure the peak bandwidth of all settings. In MEM.BW, DRAM delivers best bandwidth performance, and PCM provides about 70% the bandwidth of DRAM. In HpMC, the HRank mode can deliver roughly the same bandwidth as PCM while PCache delivers the least bandwidth. This is because PCache limits access only to the 8GB of DRAM and much of DRAM bandwidth is used for data migration. HRank instead allows the processor to directly access the PCM and thus provides extra bandwidth. We observe the same phenomenon in pF3D, AMGmk, Graph500 and UMTmk. In UMTmk and Graph500, we find that HRank bandwidth is even better than DRAM. This is due to more bank conflicts in DRAM.

The middle chart in Figure 8 reports the energy consumption. We normalized the energy to DRAM. In applications with low bandwidth requirements (i.e. applications in the left), PCM consumes less energy than DRAM due to less static power dissipation. When bandwidth increases (from left to right), the energy ratio of PCM to DRAM also increases from 0.59 (CNS.STENCIL) to 1.92 (MEM.BW), because PCM uses more dynamic write power than DRAM. PCache and HRank conserve more energy than traditional DRAM in all cases except MEM.BW. The energy saving ranges from 13% to 45%. PCache conserves more energy than HRank in MEM.BW, pF3D, MILCmk, UMTmk and CNS.STENCIL while HRank conserves more in LULESH, AMGmk and Graph500. EaC dynamically chooses the low energy policies to optimize energy while achieving almost the same bandwidth. However, EaC may still sacrifice performance in certain workloads (e.g. MEM.BW and pF3D).

The bottom chart in Figure 8 reports the latency of all memory systems. *.1LM and *.2LM represent the DRAM and PCM latency of the HpMC using three modes. We model the latency of memory systems without taking the cost of MC migration and OS updates into consideration becasue they are not always on the critical path of memory accesses. In applications with low and moderate bandwidth requirements (CNS.STENCIL to AMGmk), we observe that the latency of the three HpMC modes is similar to that of the DRAM system. In applications with high bandwidth requirements (LULESH, PF3D and MEM.BW), we observe that the PCM latency in the three modes of HpMC is higher than DRAM. This is inevitable because the PCM is highly utilized. However, the latency can be hidden by the high level of concurrency in the processor.

6. RELATED WORK

Mogul et al. [13] considered combining NAND Flash and DRAM in main memory to reduce main memory power consumption. More recent works [8, 18, 27, 17] focused on using PCM to partially or completely replace DRAM due to its more promising performance characteristics than Flash. Because page migration is the key to energy conservation and performance for main memory systems, several works have been proposed to address this problem. Huang et al. [6] proposed an OS-controlled, power-aware virtual memory to periodically migrate pages based on reference bits. Although previous OS-based approaches can improve energy consumption, OS latency is still a major concern. Several hardware-controlled systems have been studied in the literature [4, 14]. Pandey et al. [14] used the access pattern in workloads by clustering frequently accessed pages in a small subset of the memory chips to improve locality and energy consumption. Dong et al. [4] implemented an address translation mechanism in the memory controller that can dynamically migrate data between on-package and off-package memories. These works study the performance and energy with a single heterogeneous memory organization and a single management policy. Qureshi et al. [16] proposed the Dynamic Insertion Policy (DIP), which dynamically switches between LRU and BIP (Bimodal Insertion Policy) policies based on the small working set analysis. This research is close to our work, however, the proposed policy only considers a hierarchical, inclusive system without considering the potential benefits of a flat system design enabling direct access to the 2LM. In our work we do not assume that a single policy or memory organization can meet the demands of multiple applications. Furthermore, a single policy may not provide the best performance or energy efficiency within an application because of the different computational requirements of an application's phase. This motivates our investigation for switching policies dynamically based on the temporal locality of applications and how locality changes over time.

7. SUMMARY AND CONCLUSIONS

In this paper, we study different memory organizations and management policies for emerging multi-level memory systems. We evaluate the impact of several policies on the performance and energy consumption of a number of codes

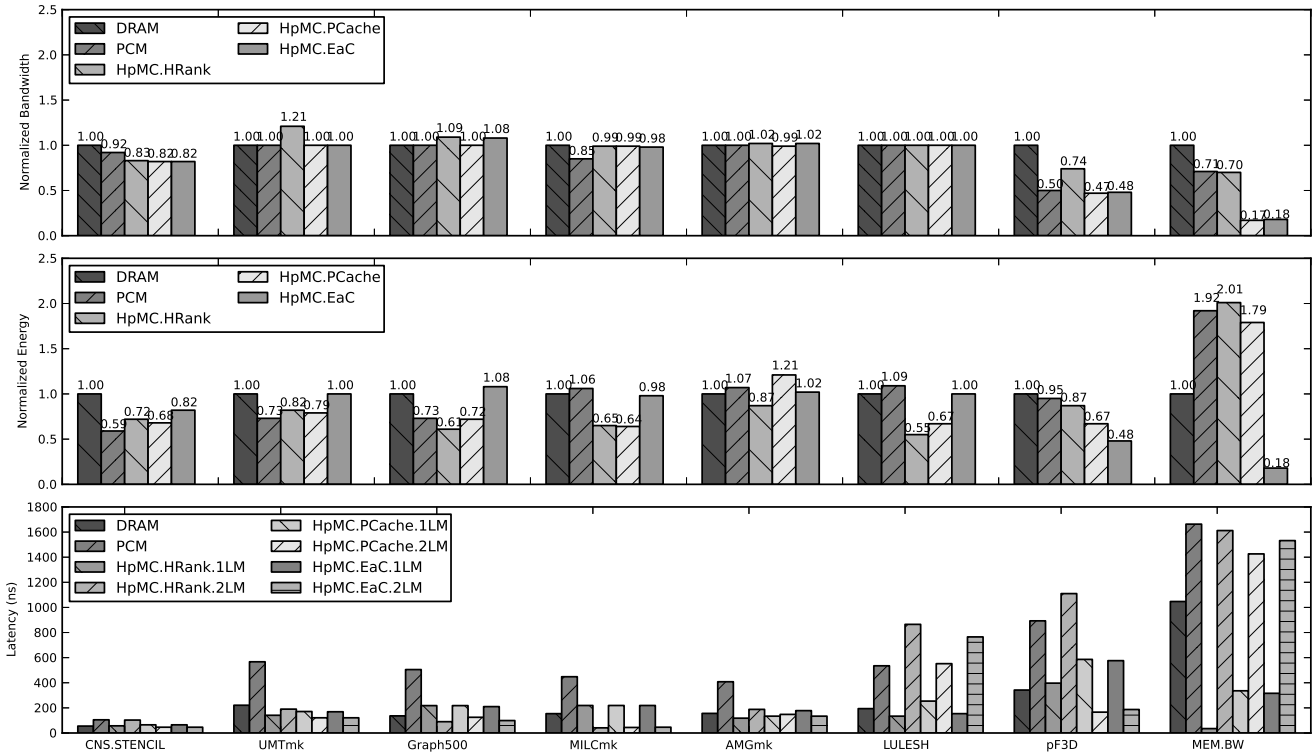

Figure 8: Memory bandwidth, energy, and latency comparison of DRAM, PCM, and three modes in HpMC.

of interest to the U.S. Department of Energy. To perform this evaluation we faced two challenges: a shortage of available tools to simulate the performance and power of applications in a multi-level memory system and a lack of guidelines to help application developers use multi-level memories more efficiently.

To overcome these challenges we created HMsim, a simulation infrastructure that provides detailed memory simulations and enables the study of n-level, heterogeneous memories. HMsim achieves this by decoupling the memory and processor simulation and leveraging existing cycle-accurate memory simulators. Our validation with existing architectures show that HMsim closely follows the same changes in performance and power. In this work, we configured HMsim with DRAM in the first level of memory and PCM in the second level. For the latter, we developed a model based on an existing DRAMsim model.

We proposed and evaluated HpMC, a new memory controller design that dynamically selects between two well-known, two-level memory policies to improve the energy efficiency of applications on heterogeneous memory systems. A key observation that led to the design of HpMC is that neither policy provides the best energy efficiency across a number of HPC applications. Furthermore, the choice of best policy changes as an application transitions between different code regions. We found that temporal locality of an application is a good indicator to determine, at run time, the memory management policy.

The results from our suite of HPC applications and micro-benchmarks demonstrate that HpMC reduces energy consumption from 13% to 45% compared to HRank and PCache, while providing almost the same memory bandwidth and significantly higher capacity than a DRAM-only system. Our hybrid system leverages the benefits of each policy by dynamically adapting to the characteristics of applications, particularly temporal locality.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-CONF-662474. AMD, the AMD Arrow logo, the AMD SimNow simulator, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.This work has been supported in part by the European Commission under Grant Agreement 610509 (NanoStreams). This material is based upon work supported in part by the National Science Foundation under Grant No. 1422788, 0910784 and 0905187.

8. REFERENCES

[1] TN-41-01: Calculating memory system power for DDR3s. Technical report, Micron Technology, Inc, December 2009.

[2] E. Chen, D. Apalkov, Z. Diao, A. Driskill-Smith, D. Druist, D. Lottis, V. Nikitin, X. Tang, S. Watts, S. Wang, S. Wolf, A. W. Ghosh, J. Lu, S. J. Poon, M. Stan, W. Butler, S. Gupta, C. K. A. Mewes, T. Mewes, and P. Visscher. Advances and Future

Prospects of Spin-Transfer Torque Random Access Memory. *Magnetics, IEEE Transactions on*, 46(6), June 2010.

[3] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. RAPL: Memory Power Estimation and Capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10.

[4] X. Dong, Y. Xie, N. Muralimanohar, and N. Jouppi. Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support. In *High Performance Computing, Networking, Storage and Analysis (SC10)*.

[5] M. Giampapa, T. Gooding, T. Inglett, and R. W. Wisniewski. Experiences with a lightweight supercomputer kernel: Lessons learned from Blue Gene's CNK. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'10, New Orleans, LA, Nov. 2010. ACM/IEEE.

[6] H. Huang, P. Pillai, and K. G. Shin. Design and Implementation of Power-aware Virtual Memory. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '03. USENIX Association.

[7] Y. Jiang, E. Z. Zhang, K. Tian, and X. Shen. Is Reuse Distance Applicable to Data Locality Analysis on Chip Multiprocessors? In *Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction*, CC'10/ETAPS'10.

[8] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable DRAM alternative. *SIGARCH Comput. Archit. News*, 37(3), June 2009.

[9] X. Liao, L. Xiao, C. Yang, and Y. Lu. Milkyway-2 supercomputer: system and application. *Frontiers of Computer Science*, 8(3), 2014.

[10] G. H. Loh and M. D. Hill. Efficiently enabling conventional block sizes for very large die-stacked dram caches. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44. ACM, 2011.

[11] L. McVoy and C. Staelin. Lmbench: Portable Tools for Performance Analysis. In *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*.

[12] J. Meza and J. Li. Evaluating row buffer locality in future non-volatile main memories. Technical report, Computer Architecture Lab, Carnegie Mellon University, December 2012.

[13] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi. Operating System Support for NVM+DRAM Hybrid Main Memory. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems*, HotOS'09. USENIX Association.

[14] V. Pandey, W. Jiang, Y. Zhou, and R. Bianchini. DMA-Aware Memory Energy Management. In *In Proceedings of HPCA*, 2006.

[15] M. Qureshi, M. Franceschini, A. Jagmohan, and L. Lastras. PreSET: Improving performance of phase change memories by exploiting asymmetry in write times. In *Computer Architecture (ISCA), 2012*, June.

[16] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely, and J. Emer. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 381–391, New York, NY, USA, 2007. ACM.

[17] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09. ACM.

[18] L. E. Ramos, E. Gorbatov, and R. Bianchini. Page Placement in Hybrid Memory Systems. In *Proceedings of the International Conference on Supercomputing*, ICS '11. ACM.

[19] R. Riesen, R. Brightwell, P. G. Bridges, T. Hudson, A. B. Maccabe, P. M. Widener, and K. Ferreira. Designing and implementing lightweight kernels for capability computing. *Concurrency and Computation: Practice and Experience*, 21(6):793–817, Apr. 2009.

[20] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. DRAMSim2: A Cycle Accurate Memory System Simulator. *Computer Architecture Letters*, 10(1), jan.-june 2011.

[21] X. Shen, J. Shaw, B. Meeker, and C. Ding. Locality Approximation Using Time. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07. ACM.

[22] J. Treibig, G. Hager, and G. Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*.

[23] H. Vandierendonck, A. Hassan, and D. Nikolopoulos. On The Energy-Efficiency of Byte-Addressable Non-Volatile Memory. *Computer Architecture Letters*, PP(99), 2014.

[24] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely. Quantifying Locality In The Memory Access Patterns of HPC Applications. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society.

[25] H. S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. Phase Change Memory. *Proceedings of the IEEE*, 98(12), Dec 2010.

[26] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu. A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme. In *Circuits and Systems, 2007. ISCAS 2007.*

[27] W. Zhang and T. Li. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In *Parallel Architectures and Compilation Techniques, 2009. PACT '09.*

[28] Y. Zhou, J. Philbin, and K. Li. The multi-queue replacement algorithm for second level buffer caches. In *USENIX Annual Technical Conference*, 2002.